



中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

实验课程: 操作系统原理实验

任课教师: 陈鹏飞

实验题目:

专业名称: 计算机科学与技术

学生姓名: 张玉瑶

学生学号: 23336316

实验地点: 实验中心 B203

实验时间: 2025.3.2

Section 1 实验概述

本实验目的：熟悉现有 Linux 内核的编译过程和启动过程，并在自行编译内核的基础上构建简单应用并启动。利用精简的 Busybox 工具集构建简单的 OS，熟悉现代操作系统的构建过程。熟悉编译环境、相关工具集，并能够实现内核远程调试。具体内容如下。

1. 搭建 OS 内核开发环境包括：代码编辑环境、编译环境、运行环境、调试环境等。
2. 下载并编译 i386（32 位）内核，并利用 qemu 启动内核。
3. 熟悉制作 initramfs 的方法。
4. 编写简单应用程序随内核启动运行。
5. 编译 i386 版本的 Busybox，随内核启动，构建简单的 OS。
6. 开启远程调试功能，进行调试跟踪代码运行。

Section 2 预备知识与实验环境

预备知识

1. x86 汇编语言程序设计
2. Linux 系统命令行工具
 - 熟练使用 Linux 命令行工具，如文件操作、进程管理、权限管理等。
 - 掌握常用命令如 `gcc`、`make`、`gdb`、`qemu` 等。

实验环境

1. 虚拟机版本/处理器型号
 - 虚拟机：VirtualBox
 - 处理器型号：支持 x86 架构的 CPU

2. 代码编译工具

- 使用 `gcc` 编译器。

3. 重要三方库信息

- `qemu`: 用于模拟 x86 架构的虚拟机环境（版本未明确）。
- `gdb`: 用于调试汇编和 C 代码。

Section 3 实验任务

1. 完成 Linux 内核编译

当前用户目录下创建文件夹 `lab1` 并进入。

```
mkdir ~/lab1  
cd ~/lab1
```

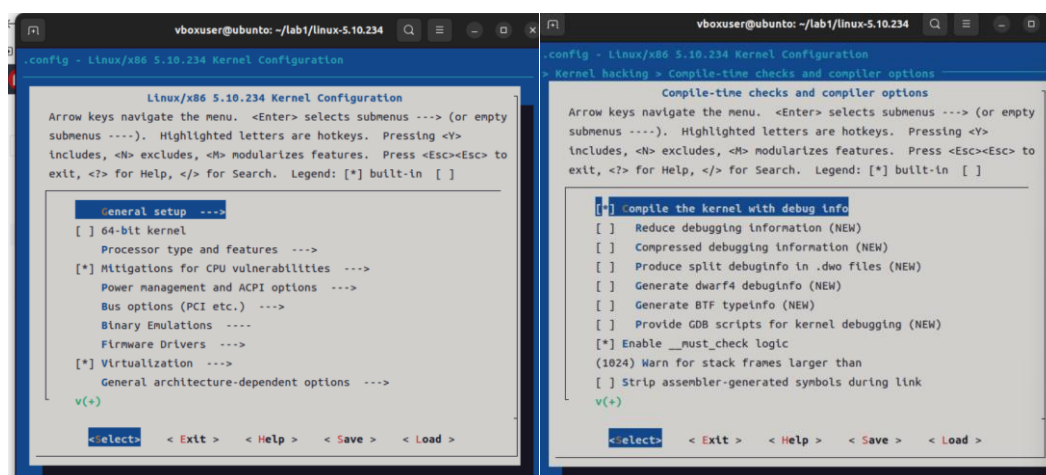
下载内核 5.10 到 `~/lab1`, 解压并进入。

```
xz -d linux-5.10.234.tar.xz  
tar -xvf linux-5.10.234.tar  
cd linux-5.10.234
```

将内核编译成 i386 32 位版本。

```
make i386_defconfig  
make menuconfig
```

在打开的图像界面中依次选择 `Kernel hacking`、`Compile-time checks and compiler options`, 最后在 `[] Compile the kernel with debug info` 输入 `Y` 勾选, 保存退出。



编译内核。

```
make -j8
```

检查 Linux 压缩镜像 `linux-5.10.234/arch/x86/boot/bzImage` 和符号表 `linux-5.10.234/vmlinux` 已经生成。

```
vboxuser@ubuntu:~$ cd ~/lab1
vboxuser@ubuntu:~/lab1$ cd linux-5.10.234
vboxuser@ubuntu:~/lab1/linux-5.10.234$ ls arch/x86/boot/bzImage
arch/x86/boot/bzImage
vboxuser@ubuntu:~/lab1/linux-5.10.234$ ls vmlinux
vmlinux
vboxuser@ubuntu:~/lab1/linux-5.10.234$
```

2. 完成 initramfs 的制作过程

下面的过程在文件夹 `~/lab1` 下进行。

```
cd ~/lab1
```

在前面调试内核中，我们已经准备了一个 Linux 启动环境，但是缺少 `initramfs`。我们可以做一个最简单的 Hello World `initramfs`，来直观地理解 `initramfs`，Hello World 程序如下。

```

GNU nano 6.2                                helloworld.c *
#include <stdio.h>

void main()
{
    printf("lab1: Hello World\n");
    fflush(stdout);
    /* 让程序打印完后继续维持在用户态 */
    while(1);
}

```

用 cpio 打包 initramfs。

```
echo helloworld | cpio -o --format=newc > hwinitramfs
```

启动内核，并加载 initramfs。

```
qemu-system-i386 -kernel linux-5.10.234/arch/x86/boot/bzImage -initrd
hwinitramfs -s
```

调试 gdb，可以看到 gdb 中输出了 lab1: Hello World\n

```

vboxuser@ubuntu:~$ cd ~/lab1
vboxuser@ubuntu:~/lab1$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs.html>
[ Legend: Modified register | Code | Heap | Stack | String ]

-- registers --
[!] Command 'registers' failed to execute properly, reason: Failed to determine
memory layout

-- stack --
[!] Command 'dereference' failed to execute properly, reason: Failed to determine
memory layout

-- code:x86:32 --
0xffea      add     BYTE PTR [eax], al
0xffec      add     BYTE PTR [eax], al
0xffee      add     BYTE PTR [eax], al
→ 0xffff0    add     BYTE PTR [eax], al
0xffff2     add     BYTE PTR [eax], al
0xffff4     add     BYTE PTR [eax], al

```

```
[*] Command 'dereference' failed to execute properly, reason: Failed to determine memory layout

code:x86:32
0xffea      add     BYTE PTR [eax], al
0xffec      add     BYTE PTR [eax], al
0xffee      add     BYTE PTR [eax], al
→ 0xffff0    add     BYTE PTR [eax], al
0xffff2     add     BYTE PTR [eax], al
0xffff4     add     BYTE PTR [eax], al
0xffff6     add     BYTE PTR [eax], al
0xffff8     add     BYTE PTR [eax], al
0xffffa     add     BYTE PTR [eax], al

threads
[#0] Id 1, stopped 0xffff0 in ?? (), reason: SIGTRAP

trace
[#0] 0xffff0 →add BYTE PTR [eax], al

Python Exception <class 'gdb.error'>: Remote I/O error: Function not implemented
Error while executing Python code.
(remote) gef> break start_kernel
Breakpoint 1 at 0xc21537c3: file init/main.c, line 849.
(remote) gef> c
Continuing.
```

```
[ 2.657153] Freeing unused kernel image (initramfs) at 0000000000000000
P[ 2.664040] Write protecting kernel text and data
E[ 2.664848] Run helloworld as init process
(lab1: Hello World
B[ 3.070730] input: ImExPS/2 Generic Explorer
(l
```

3. 完成内核的装载和启动过程

使用 `qemu` 启动内核并开启远程调试。

```
qemu-system-i386 -kernel linux-5.10.234/arch/x86/boot/bzImage -s -S -
append "console=ttyS0" -nographic
```

运行的 Linux 系统能成功启动，并且最终以 Kernel panic 宣告结束。

```
[ 2.349544] mount_root+0xd9/0xf1
[ 2.349626] prepare_namespace+0x116/0x141
[ 2.349718] kernel_init_freeable+0x1be/0x1cb
[ 2.349817] ? rest_init+0x92/0x92
[ 2.349900] kernel_init+0x8/0xde
[ 2.349979] ret_from_fork+0x1c/0x28
[ 2.351026] Kernel Offset: disabled
[ 2.351450] ---[ end Kernel panic - not syncing: VFS: Unable to mount root f-
```

4. 完成 Busybox 的编译、启动过程

下面的过程在文件夹~/lab1 下进行。

```
cd ~/lab1
```

从课程网站处下载下载 Busybox 到~/lab1，然后解压。

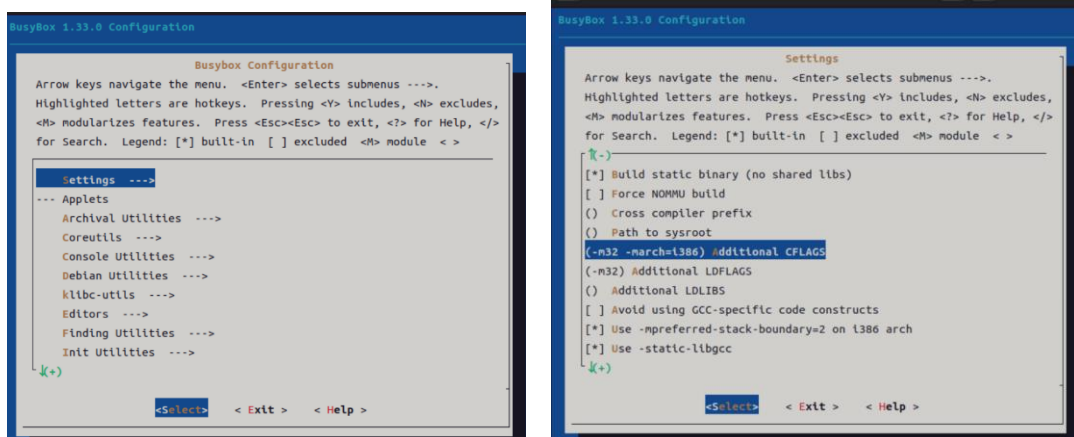
```
tar -xf Busybox_1_33_0.tar.gz
```

编译：

```
make defconfig
```

```
make menuconfig
```

进入 settings，然后在 Build BusyBox as a static binary(no shared libs)处输入 Y 勾选，然后分别设置() Additional CFLAGS 和() Additional LDFLAGS 为(-m32 -march=i386) Additional CFLAGS 和(-m32) Additional LDFLAGS。



保存退出，然后编译。

```
make -j8
```

```
make install
```

4. 完成 Busybox 的远程调试

将安装在_install1 目录下的文件和目录取出放在~/lab1/mybusybox 处。

```
cd ~/lab1
```

```
mkdir mybusybox
```

```
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
```

```
cp -av busybox-1_33_0/_install/* mybusybox/
```

```
cd mybusybox
```

initramfs 需要一个 init 程序，可以写一个简单的 shell 脚本作为 init。
用 gedit 打开文件 `init`，复制入如下内容，保存退出。

```
mount -t proc none /proc
mount -t sysfs none /sys
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
exec /bin/sh
```

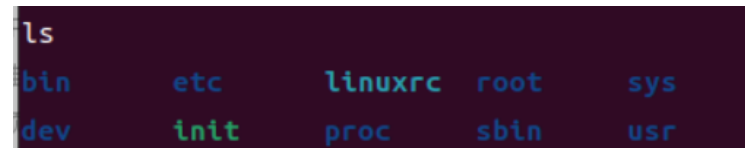
加上执行权限。

```
chmod u+x init
```

将 x86-busybox 下面的内容打包归档成 cpio 文件，以供 Linux 内核做
initramfs 启动执行。

```
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
~/lab1/initramfs-busybox-x86.cpio.gz
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd
initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0"
```

然后使用 `ls` 命令即可看到当前文件夹。



```
ls
bin      etc      linuxrc  root     sys
dev      init     proc     sbin     usr
```

7. Linux 0.11 内核的编译、启动和调试


```

vboxuser@ubuntu:~/lab1$ cd Linux-0.11
vboxuser@ubuntu:~/lab1/Linux-0.11$ pwd
/home/vboxuser/lab1/Linux-0.11
vboxuser@ubuntu:~/lab1/Linux-0.11$ ls
boot  hdc-0.11.img  init  lib  Makefile.header  README.md  tools
fs    include  kernel  Makefile  mm  readme.old
vboxuser@ubuntu:~/lab1/Linux-0.11$ sed -i 's/CFLAGS += $(RAMDISK) -Iinclude
AGS += $(RAMDISK) -Iinclude -g -m32/' Makefile
vboxuser@ubuntu:~/lab1/Linux-0.11$ sed -i 's/LDFLAGS += -Ttext 0 -e startup
LDFLAGS += -Ttext 0 -e startup_32 -m elf_i386/' Makefile
vboxuser@ubuntu:~/lab1/Linux-0.11$ cat Makefile

```

找到 Makefile，查看里面的内容，通过 make 工具进行编译；另外，因为需要调试，所以需要在 gcc 编译命令中添加 -g 参数，产生内核的符号表；编译 32 位版本的内核，需要添加 -m32 参数；

```

LDFLAGS += -Ttext 0 -e startup_32 -m elf_i386
CFLAGS += $(RAMDISK) -Iinclude -g -m32

```

启动 gdb； 加载 linux 0.11 的符号表（一般位于 tools/system）； target remote:1234 远程连接 qemu 调试； 设置源码目录： directory linux0.11 的源码路径（请自行替换）； 设置汇编代码的形式： set disassembly-flavor intel 在关键位置设置断点如在地址 0x7c00、内核入口函数（main）等。

```

vboxuser@ubuntu:~$ cd lab1
vboxuser@ubuntu:~/lab1$ cd Linux-0.11
vboxuser@ubuntu:~/lab1/Linux-0.11$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
[ Legend: Modified register | Code | Heap | Stack | String ]

----- registers -----
[!] Command 'registers' failed to execute properly, reason: Failed to deter
memory layout

----- stack -----
[!] Command 'dereference' failed to execute properly, reason: Failed to det
ne memory layout

----- code:x86:32 -----
0xffea <do_execve+01c6> add    BYTE PTR [eax], al
0xffec <do_execve+01c8> add    BYTE PTR [eax], al
0xffee <do_execve+01ca> add    BYTE PTR [eax], al

209         i = inode->i_mode;
210         e_uid = (i & S_ISUID) ? inode->i_uid : current->euid;
211         e_gid = (i & S_ISGID) ? inode->i_gid : current->egid;
212         if (current->euid == inode->i_uid)
213             i >>= 6;
→ 214         else if (current->egid == inode->i_gid)
215             i >>= 3;
216         if (!(i & 1) &&
217             !((inode->i_mode & 0111) && suser())) {
218             retval = -ENOEXEC;
219             goto exec_error2;

----- threads -----
[#0] Id 1, stopped 0xffff0 in do_execve (), reason: SIGTRAP

----- trace -----
[#0] 0xffff0 → do_execve(eip=0x0 <startup_32>, tmp=0x0, filename=0x0 <startu
>, argv=0x0 <startup_32>, envp=0x0 <startup_32>)
[#1] 0x0 → <startup_32+0> add BYTE PTR [eax], al

Python Exception <class 'gdb.error': Remote I/O error: Function not imple

```

找到 hdc-0.11.img 硬盘镜像文件，这是 linux 0.11 操作系统启动后的根文件系统，相当于在 qemu 虚拟机里装载的硬盘；

先用 fdisk 命令查看磁盘的分区情况以及文件类型(minix):

```

vboxuser@ubuntu:~/lab1/Linux-0.11$ fdisk -l hdc-0.11.img
Disk hdc-0.11.img: 59.55 MiB, 62447616 bytes, 121968 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device            Boot Start    End Sectors  Size Id Type

```

创建本地挂载目录。

```
mkdir hdc
```

显示盘空间。

```
df -h
```

挂载 linux 0.11 硬盘镜像。

```
sudo mount -t minix -o loop,offset=512 hdc-0.11.img hdc (注意是 hdc 的完整路径)
```

查看是否挂载成功。

```
df -h (在 ubuntu22.04 中执行，查看是否出现挂载的 hdc 路径)
```

```
vboxuser@ubuntu:~/lab1/Linux-0.11$ mkdir hdc
vboxuser@ubuntu:~/lab1/Linux-0.11$ sudo mount -t minix -o loop,offset=512 hdc-0.11.img hdc
vboxuser@ubuntu:~/lab1/Linux-0.11$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
tmpfs	392M	1.7M	390M	1%	/run
/dev/sda3	42G	23G	18G	57%	/
tmpfs	2.0G	0	2.0G	0%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	2.0G	0	2.0G	0%	/run/qemu
/dev/sda2	512M	6.1M	506M	2%	/boot/efi
tmpfs	392M	136K	392M	1%	/run/user/1000
/dev/sr0	58M	58M	0	100%	/media/vboxuser/VBox_GAs_7.1.6
/dev/loop17	58M	13M	46M	23%	/home/vboxuser/lab1/Linux-0.11/hdc

查看挂载后的 hdc 目录结构

```
ll hdc
```

```
vboxuser@ubuntu:~/lab1/Linux-0.11$ ll hdc
total 13
drwxr-xr-x 10 root      root      176 3月 22  2004 ./
drwxrwxr-x 12 vboxuser vboxuser 4096 3月  5 15:59 ../
drwxr-xr-x  2 root      root      912 3月 22  2004 bin/
drwxr-xr-x  2 root      root      336 3月 22  2004 dev/
drwxr-xr-x  2 root      root      224 3月 22  2004 etc/
drwxr-xr-x  8 root      root      128 3月 22  2004 image/
drwxr-xr-x  2 root      root        32 3月 22  2004 mnt/
drwxr-xr-x  2 root      root        64 3月 22  2004 tmp/
drwxr-xr-x 10 root      root      192 3月 30  2004 usr/
drwxr-xr-x  2 root      root        32 3月 22  2004 var/
```

进入 hdc 的 usr 目录。

```
cd hdc/usr
sudo touch hello.txt
sudo vim hello.txt （编辑文件）
```

卸载文件系统 hdc。

```
sudo umount /dev/loop （查看具体的 loop 设备）
df -h （查看是否已经卸载）
```

```
vboxuser@ubuntu:~/lab1/Linux-0.11$ sudo umount /dev/loop17
[sudo] password for vboxuser:
vboxuser@ubuntu:~/lab1/Linux-0.11$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           392M  1.7M  390M   1% /run
/dev/sda3       42G   23G   18G  57% /
tmpfs           2.0G    0  2.0G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.0G    0  2.0G   0% /run/qemu
/dev/sda2       512M   6.1M  506M   2% /boot/efi
tmpfs           392M  140K  392M   1% /run/user/1000
/dev/sr0        58M   58M    0 100% /media/vboxuser/VBox_GAs_7.1.6
```

重新用 qemu 启动 linux 0.11

观察/usr 目录下是否有 hello.txt 文件

```
74501 ~ # ls
[/usr/root]# ls
README      gcclib140  hello      hello.c    mtools.howto
```

Section 5 实验总结与心得体会

这是一次全新的实验，对我来说也是前所未有的挑战。以前从未接触过这个系统，也从未在终端中编写过命令，完全是一步步跟随朋友和 DeepSeek 的指引前行。经历了无数次的删除、编译和重新编译，虽然最终仍未完全理清其中的奥秘，但这段经历让我深刻意识到，自己还需要更加努力地学习。每一次尝试都是一次成长，未来我会继续探索，直到彻底掌握这些知识。